Page 1, Lines 30-33; Page 2, Lines 1-9

When computer code written in Java executes in what is referred to as a Java execution environment, it is desirable to have a class location system that can quickly examine the class path for the location of a given class specified by a name of the class. This is the only use for the class path during execution. No other types of searching, such as enumeration of all classes in a particular package, is necessary at execution time. Also, it is not a concern if an unsuccessful search for a class takes a long time, since at execution time this results in an error situation. A class locator service, known as a JPS, or the Java Package Service, exposes an API which describes the class path as a single class source, with methods such as FindClass and EnumClasses regardless of the current configuration of the class path. These methods find all classes with a fiven name in all or a specified package, and enumerate all classes of a given name respectively.

## REMARKS

### *Status of the Application*

Upon entry of this amendment, the Specification will have been amended to correct typographical errors in the originally filed application. Examiner has suggested to Applicant to amend the specification to correct what appears to Examiner to have been a typographical error. Applicant respectfully submits that the "As-Filed" application does not possess the aforementioned error, but, nevertheless offers an amended Specification to comply with the

Examiner's suggestion. Specifically, the changes to the Specification are ***merely ministerial and do not add new matter***. With respect to the Claims of the present application, Claims 1-24 remain pending in this case. Claims 1,2 and 10-14 stand rejected under 35 U.S.C. §103 (a) as allegedly being obvious over U.S. Patent 5,573,574 (*Elko et al.*). Claims 5-9, and 15-24 stand rejected under 35 U.S.C. §103(a) as allegedly being obvious over U.S. Patent 6,230,284 B1 (*White et al.)* in view of *Elko et al.*.

In view of the foregoing amendments and following remarks, Applicant respectfully requests reconsideration of the present application and an early Notice of Allowance.

### *35 U.S.C. § 103(a) Rejections*

*Prima Facie* Obviousness

To establish a *prima facie* case of obviousness, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art to modify the reference or to combine reference teachings. Further, there must be a reasonable expectation of success after combining the references the intended purpose of the invention is realized. **Lastly, the prior art reference (or references when combined) must teach or suggest all the claim limitations.** The teaching or suggestion to make the claimed combination and the reasonable expectation of success must both be found in the prior art, and not based on applicant's disclosure. *In re Vaeck*, 947 F.2d 488, 20 U.S.P.Q.2d 1438 (Fed. Cir. 1991). Applicant respectfully submits that a *prima facie* case of obviousness has not

been made for claims 1-24 of the present application.

The Present Invention

The present invention contemplates a system and methods for optimizing the caching of classes. In one aspect of the invention, **only** ***selected*** elements in a class path are cached. These elements are ***selected*** as those that have associated mechanisms that provide notification of changes. In this manner, the integrity of the cache can be maintained with little overhead involved in the independent tracking of the class path elements. By keeping separate caches, reconstruction of a cache due to a detected change is limited to that cache only. Specifically, the present invention, as claimed, *inter alia* is operable to locate classes in a class path by ***generating a cache*** of information relating to the classes in the class path, ***requesting a search*** of the class path, and ***searching the cache*** to satisfy the search request. Also, the present invention is capable of, *inter alia*, locating classes in a multi-element class path by generating a search request for desired requests within the multi-element class path, and ***independently*** satisfying the request in association with each element in the class ***such that at least one of the elements has a cache of information sufficient to satisfy the request for that element.*** Further, the present invention contemplates, *inter alia*, creating caches for selected elements of a class path. In this context, the present invention parses the class path into names of elements, determines that elements are viable for caching, and creates caches for the viable elements. These contemplated functions are operable by a system as offered by the present invention having a class path

manager that receives request for identification or enumeration of classes in a class path, a cache to cache a viable element of the class path, and a wrapper that receives requests (e.g. identification or enumeration of classes in the class path) from the class path manager and provides a transparent level of indirection to services that are specific to the cache viable element.

Differences from *Elko et al.*

*Elko et al.* disclose a system and method for controlling coherence of data elements sharable among a plurality of independently-operating CPCs (central processing complexes) in a multi-system complex (a.k.a. a parallel sysplex). The multi-system complex contains sysplex DASDs (direct access storage devices) and a high-speed SES (shared electronic storage) facility. In operation, sysplex shared data elements are stored in the sysplex DASD under a unique sysplex data element name, which is used for sysplex coherence control. In this system, any CPC may copy any sysplex data element into a local cache buffers (LCB) in the CPC's main storage, where it has an associated sysplex validity bit. The copying CPC executes a sysplex coherence registration command which request a SES processor to verify that the data element name is already in the SES cache, and to store the name of the data element in a SES cache entry if found in the SES cache. This registration command communicates to SES the CPC location of the validity bit for the LCB containing that data element copy. Each time another copy of the data element is stored in any CPC LCB, a registration command is executed to store the location of that copy's CPC validity bit into a local cache register (LCR) associated with its data element

name. In this manner, each LCR accumulates all CPC location for all LCB validity bits for all valid copies of the associated data element in the sysplex – for maintaining data coherence throughout the sysplex. (*Elko et al.* ABSTRACT).

In contrast to the inventions of Applicant's claims 1-24, however, the systems and methods taught by *Elko et al.* **do not teach a system or methods that contemplate the creation of cache having information relating to the classes of a class path, requesting a search on the class path, and searching the created cache to satisfy the search request.** (See Claims 1-24 of the present application). Rather, *Elko et al.* teach a system and method that provides data coherence among cooperating elements of a multiple processing environment where more than one copy of a data element may persist. *Elko et al.* goes further to teach the use of a LCR (local cache register) as the mechanism by which these data element copies are tracked.

Examiner suggests that the *Elko* SES cache is equivalent to a class path cache. (Page 3 of the Current Action). Applicant respectfully disagrees with the Examiner's broad-stroke analogy. Specifically, Applicant submits **that a class path is a very specific construct in space of software development and as such maintains very particular characteristics-** ***characteristics that are not implicitly or explicitly taught by Elko et al.*** Contrary to Examiner's suggestion, it is not plausible that one of ordinary skill in the art turn to *Elko et al.* to find teachings related to the creation and searching of a class path cache.

***Elko et al. simply do not teach or even suggest the creation of a cache contained therein information relating to the class of a class path, the requesting and execution of***

***search on the cache to locate a particular class.***

As *Elko et al.* do not teach every limitation of the present invention, a *prima facie* case of obviousness has not been made. Moreover, Applicant submits that the present invention is not obvious over the *Elko et al.* reference when combined with the other herein described cited references since there is no motivation in the *Elko et al.* reference to combine with the other herein described references. *Elko et al.* is directed to a system and methods for controlling coherence of data elements in a distributed processing environment. (*Elko et al.* ABSTRACT). The present invention, as stated, is directed to optimizing class identification for a class path. One of ordinary skill in the art would find it difficult to, refer to, yet alone, combine the teachings found in *Elko et al.* with the other herein described references to teach the present invention. Further, even if the *Elko et al.* reference is combined with the other herein described references, the present invention is not obvious over these herein described references since these herein described references, alone, or in combination **fail to teach every limitation of the present invention**. For these reasons, Applicant respectfully requests that the obviousness rejection be withdrawn.

Differences from *White et al.*

*White et al.* disclose a system and methods to determine what files are needed to optimally execute a computer program in a desired state. In operation, the files needed to execute the computer program until it reaches a desired state are identified and executed on the

computer on which it resides or it can be transmitted from one computer to another (e.g. from a server to a client computer). The optimized file enables a computer to execute a computer program until a previously determined desired state is attained. Other files may be provided which enables the computer with the ability to execute the computer program beyond the desired state. The desired state of the computer program may be, for example, the point where user input can be accepted by the computer program. (*White et al.* Col. 7, Lines 1-15).

In contrast to the inventions of Applicant's claims 1-24, however, the systems and methods taught by *White et al.* **do not teach a system or methods that contemplate the creation of cache having information relating to the classes of a class path, requesting a search on the class path, and searching the created cache to satisfy the search request.** (See Claims 1-24 of the present application). Rather, *White et al.* teach a system and method for aggregating components of a computer program into a operational/executable form to bring the operating computer into a desired state. (*White et al.* ABSTRACT).

Examiner suggests that *White et al.* discloses a multi-element class path. (Page 6 of the Current Action). Applicant does not disagree with the Examiner's suggestion. However, Applicant submits, and as Examiner agrees, that *White et al.* fail to teach the steps of generating a search request on a class path, and independently satisfying the request in association with each element in the class path, wherein at least one of the elements has cache information sufficient to satisfy the request for that element. Examiner suggests that *Elko et al.* teachings, when combined with *White et al.*, make up for the *White et al.* disclosure deficiencies, to teach the present

invention. Applicant respectfully disagrees with the Examiner's suggestion as *Elko et al.*, as indicated above, do not offer any inherent or explicit disclosure that come close to teaching the present invention.

***White et al. simply do not teach or even suggest the creation of a cache contained therein information relating to the class of a class path, the requesting and execution of search on the cache to locate a particular class.***

As *White et al.* do not teach every limitation of the present invention, a *prima facie* case of obviousness has not been made. Moreover, Applicant submits that the present invention is not obvious over the *White et al.* reference when combined with the other herein described cited references since there is no motivation in the *White et al.* reference to combine with the other herein described references. *White et al.* is directed to a system and methods for optimizing the execution of a computer program operating on a computer. The present invention, as stated, is directed to optimizing class identification for a class path. One of ordinary skill in the art would find it difficult to, refer to, yet alone, combine the teachings found in *White et al.* with the other herein described references to teach the present invention. Further, even if the *White et al.* reference is combined with the other herein described references, the present invention is not obvious over these herein described references since these herein described references, alone, or in combination **fail to teach every limitation of the present invention**. For these reasons, Applicant respectfully requests that the obviousness rejection be withdrawn.

**Claim Analysis:**

*Independent Claims 1 and 10:*

Examiner has rejected independent claims 1 and 10 as being obvious over *Elko et al.* Examiner suggests that *Elko et al.* discloses a method of generating a cache of information relating to the classes, requesting a search of the class path, and searching the cache to satisfy the requested search that teaches the use of a class files database teach the present invention as claimed. However, as described above, *Elko et al.* do not teach every limitation found in claims 1 and 10. ***Specifically, Elko et al. do not teach the creation of cache of having information pertinent to classes of a class path.*** Accordingly, a *prima facie* case of obviousness has not been made. Applicant respectfully requests that the obviousness rejection be withdrawn for these claims on this basis.

*Independent Claims 5, 15, 17, 22, and 23:*

Examiner has rejected independent claims 5, 15, 17, 22, and 23 as being obvious over *Elko et al.* in view of *White et al.* Examiner suggests that *Elko et al.* discloses a method of generating a cache of information relating to the classes, requesting a search of the class path, and searching the cache to satisfy the requested search, and when combined with *White et al.*, that the Examiner suggests teaches the creating of searchable cache multi-class class path, teach the present invention as claimed. However, as described above, *Elko et al.*, and *White et al.*, alone, or when combined do not teach every limitation found in claims 1, 5, 15, 17, 22, and 23. ***Specifically, neither, Elko et al., and White et al., alone, or in combination teach the creation***

***of a cache having information pertinent to classes of a class path on which a search operates.***

Accordingly, a *prima facie* case of obviousness has not been made. Applicant respectfully requests that the obviousness rejection be withdrawn for these claims on this basis.

*Dependent Claims 2-4, 6-9, 11-14, 16, 18-21 and 24:*

Examiner has rejected dependent claims 2-4, 6-9, 11-14, 16, 18-21, and 24 as being allegedly obvious over *Elko et al.*, and *White et al.* for reasons set forth in the present action. Inasmuch as dependent claims 2-4, 6-9, 11-14, 16, 18-21, and 24 depend either directly or indirectly from independent claims 1, 5, 15, 17, 22, and 23 , Applicants respectfully submit that they too patentably define over the prior art of record for the same reasons. Nevertheless, Applicants further submit that the *Elko et al.*, and *White et al.*, references, nor any other prior art of record, whether alone or in combination, teaches or suggests the following subject matter:

*Claim 2* – computer readable medium having instructions stored thereon for causing a computer to perform the steps of generating a cache of information relating to the classes of a class path, requesting a search of the class path, and searching the cache to satisfy the requested search.

*Claim 3* – computer readable medium having instructions stored thereon for causing a computer to perform the steps of generating a cache of information relating to the classes of a class path, requesting a search of the class path, and searching the cache to satisfy the requested search wherein the class path comprises multiple elements, each element having multiple classes

stored therein.

*Claim 4* – computer readable medium having instructions stored thereon for causing a computer to perform the steps of generating a cache of information relating to the classes of a class path, requesting a search of the class path, and searching the cache to satisfy the requested search wherein at least one of the elements comprises a ZIP file.

*Claim 6* – computer readable medium having instructions stored thereon for causing a computer to perform the steps of generating a search request for desired classes within the multi-element class path and independently satisfying the request in association with each element in the class path, wherein at least one of the elements has a cache of information sufficient to satisfy the request for that element.

*Claim 7* – computer readable medium having instructions stored thereon for causing a computer to perform the steps of generating a search request for desired classes within the multi-element class path and independently satisfying the request in association with each element in the class path, wherein at least one of the elements has a cache of information sufficient to satisfy the request for that element wherein at least one of the elements comprises a ZIP file.

*Claim 8* – computer readable medium having instructions stored thereon for causing a computer to perform the steps of generating a search request for desired classes within the multi-element class path and independently satisfying the request in association with each element in the class path, wherein at least one of the elements has a cache of information sufficient to satisfy the request for that element wherein the classes comprise Java classes.

*Claim 9* – computer readable medium having instructions stored thereon for causing a computer to perform the steps of generating a search request for desired classes within the multi-element class path and independently satisfying the request in association with each element in the class path, wherein at least one of the elements has a cache of information sufficient to satisfy the request for that element wherein at least one of the elements comprises a Java Package Manager.

*Claim 11* – method of creating caches for selected elements of a class path such that the class path is parsed into names of element, a determination is made which elements are viable for caching, and caches are created for those elements which are viable, wherein the viability of an element for caching is dependent on the ease of tracking which elements have had changes in them.

*Claim 12* – method of creating caches for selected elements of a class path such that the class path is parsed into names of element, a determination is made which elements are viable for caching, and caches are created for those elements which are viable, wherein the viability of an element for caching is determined based on it being a predetermined type.

*Claim 13* – method of creating caches for selected elements of a class path such that the class path is parsed into names of element, a determination is made which elements are viable for caching, caches are created for those elements which are viable and a check is made of the registry to see if the element already had a cache associated with it.

*Claim 14* – method of creating caches for selected elements of a class path such that the

class path is parsed into names of element, a determination is made which elements are viable for caching, caches are created for those elements which are viable and a determination of an existing cache is up to date is made.

*Claim 16* – a class manager having a means for receiving requests to search a multi-element class path for classes and a means for transferring such requests through a wrapper associated with each element to invoke element specific search methods, wherein at least one such element specific search method comprises searching a cache associated with such element.

*Claim 18* – a class path manager having a means for parsing the multi-element class path into names of elements, a means for determining whether each element is a viable cache candidate, means for creating a cache for such viable candidates, and means for creating indirection wrappers for each element to map class searches to each element for independent handling wherein the cache for each viable candidate comprises a name of the class.

*Claim 19* – a class path manager having a means for parsing the multi-element class path into names of elements, a means for determining whether each element is a viable cache candidate, means for creating a cache for such viable candidates, and means for creating indirection wrappers for each element to map class searches to each element for independent handling wherein the elements are selected from the group consisting of directories, ZIP files and Java Package Manager.

*Claim 20* – a class path manager having a means for parsing the multi-element class path into names of elements, a means for determining whether each element is a viable cache candidate, means for creating a cache for such viable candidates, and means for creating indirection wrappers for each element to map class searches to each element for independent handling wherein the directories are not cached.

*Claim 21* – a class path manager having a means for parsing the multi-element class path into names of elements, a means for determining whether each element is a viable cache candidate, means for creating a cache for such viable candidates, and means for creating indirection wrappers for each element to map class searches to each element for independent handling wherein the viability of an element for caching is dependent on the ease of tracking which elements have had changes in them.

*Claim 24* – a computer readable medium having instructions stored thereon for causing a computer to perform a method of locating classes in multi-element class path such that a search request for desired classes within the multi-element class path is generated, the request in association with each element in the class path is independently satisfied, and a date/time stamp on the element having the cache of information to determine if the cache is up to date is checked, wherein at least one of the elements has a cache of information sufficient to satisfy the request for that element, wherein changes to the element result in recreation of a cache .

## CONCLUSION

For all the foregoing reasons, Applicant respectfully submits that claims 1-24 patentably define over the prior art of record. Reconsideration of the present Office Action and an early Notice of Allowance are respectfully requested.

Attached hereto is a marked-up version of the changes made to the specification and claims by the current amendment. The attached page is captioned "Version With Markings To Show Changes Made."

Respectfully submitted,

George J. Awad
Registration No. 46,528

Date: July 2, 2002

WOODCOCK WASHBURN LLP
One Liberty Place - 46th Floor
Philadelphia, PA 19103
(215) 568-3100

## VERSION WITH MARKINGS TO SHOW CHANGES MADE

**In the Specification:**

Please replace the following paragraph found at Page 1, Lines 30-33; Page 2, Lines 1-9.

When computer code written in Java executes in what is referred to as a Java execution environment, it is desirable to have a class location system that can quickly examine the class path for the location of a given class specified by a name of the class. This is the only use for the class path during execution. No other types of searching, such as enumeration of all classes in a particular package, is necessary at execution time. Also, it is not a concern if an unsuccessful search for a class takes a long time, since at execution time this results in an error situation. A class locator service, known as a JPS, or the Java Package Service, exposes an API which describes the class path as a single class source, with methods such as FindClass and EnumClasses regardless of the current configuration of the class path. These methods fin[al]d all classes with a given name in all or a specified package, and enumerate all classes of a given name respectively.